The Efficient BiDAF

Stanford CS224N Default Project

Sri Vardhamanan .A
Department of Computer Science
Stanford University
sri19189@stanford.edu

Abstract

In this project, we build a question answering system for the Stanford Question Answering Dataset (SQuAD 2.0) [1]. We start with the ever-popular **Bi-Directional Attention Flow** (BiDAF) model [2] and explore few approaches to enhance them. Specifically, we introduce three enhancements, **pre-trained character embedding** representation that is acquired via existing pre-trained Glove word embedding, A **Cross Attender** using multi-head attention that replaces the typical query-to-context and context-to-query Attention Flow Layer, and a **transformer encoder** with multi-head self-attention that replaces the Word embedding Layer and Modelling Layer. we share the model performance with each enhancement and show that our final model achieves the score of x.xx F1 score and x.xx EM score, which outperforms the traditional BiDAF model.
Our Codebase: https://github.com/Lawliet19189/The-Efficient-BiDAF

1 Introduction

For the past few years there have been a growing interest in the field of Question Answering. There are variety of tasks under Question Answering like Open-domain Question Answering, Closed-book question answering, abstractive QA extraction, Tabular QA extraction, etc. In this project we look at a specific QA task known as extractive Machine Reading Comprehension and we limit our dataset to only the Stanford Question Answering Dataset (SQuAD) 2.0. In recent years, the traditionally clever sequence-to-sequence approaches have been replaced with massive pre-trained models like BERT [3]. In this project, we look at one of the traditional approach known as Bi-Directional Attention Flow (BiDAF). BiDAF was one of the strong performing models on SQuAD 1.0 Dataset [4], which achieved SOTA but was soon replaced by other massive pre-trained models. Few of the huge bottlenecks with these traditional approaches are that they process the input sequence sequentially with the use of timestep-based networks like Recurrent Neural Network [5]. These seq-to-seq networks are non-parallelizable, occupies huge memory, and has problems with remembering the long term context. Their in-ability to make use of the transfer learning technique is also a cause for concern.

In this work we experiment with the latest advancements since the original BiDAF and we implement few of the significant advancements that are beneficial to our model (Please refer Fig. 1). Specifically, we improve the contextual embedding layer with transformer encoders [6] and pre-trained character embedding, we improve the Attention Flow Layer by introducing Multi-headed Cross Attention and finally, we replace the modelling layer with another multi-headed self-attention encoder. The transformer encoders solves the underlying problems of sequence-to-sequence networks, like slow training and inference, in-ability to capture long-term memory succinctly and in-ability to make use of transfer learning. Due to the increased capacity of learning and extensibility, we believe the Multi-headed Cross Attention would be beneficial than the traditional query-to-context and context-to-query attention. Additionally, we also implement few of the recent beneficial transformer encoder techniques like GELU [7], scale-norm [8] and position infused Attention [9] [10].

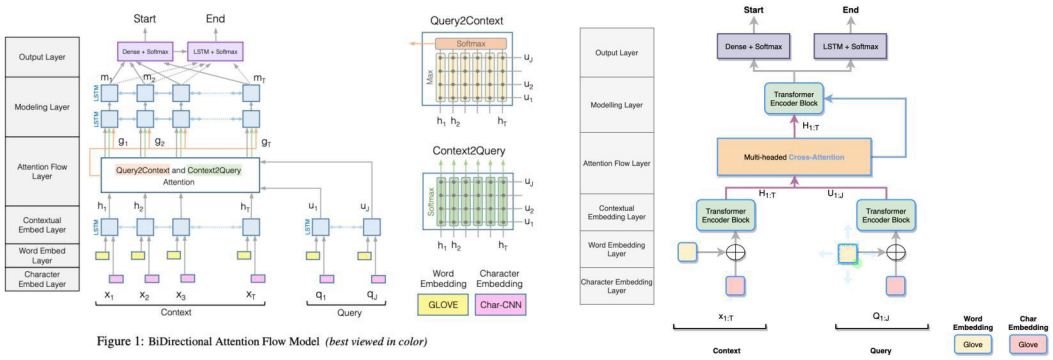

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

Figure 1: An Overview of the original BiDAF model (left) and the Overview of our model (right).

We showcase our experiments with respective metrics to signify each enhancements contribution. Our experiments shows that our model clearly outperforms the traditional approaches and also leaves room for improvement with more compute resource.

Note: Due to the limitation with time and compute resource, our experiments are generally compared with 1/4 of full training. In most-cases, this is 8 epochs with 128 batch size and 100 hidden dimension. We considered this to be okay since increase in model capacity and time would benefit transformer model more than seq-to-seq model due to efficiency and learning capability. Also, We are able to identify the pattern early whether the model is improving or not by inferring from the metric graph.

2 Related Work

Prior to the introduction of large pre-training models and transformer architectures, BiDAF introduced a strong approach to tackle reading comprehension which utilized multiple characteristics of the data (word embedding, character embedding and phrases), memory-less context-to-query and query-to-context Attention, and shallow recurrent neural networks. The model outperformed existing approaches and reached SOTA on SQuAD 1.0 leaderboard. The major disadvantage of this model was that it was non-paralleizable as it was using time-step based RNN. This resulted in huge training time and inference time.

The significant variant of this approach (non-pretraining approach) came in 2018 (post transformers) called QANet, which removed the underlying problems of BiDAF by using encoders. The model captures both local features and global features by using convolution and self-attention, respectively. As the model was non-recurrent, it was easily paralleizable and therefore performed approximately 4 times faster than the original BiDAF on training and 7 times faster on infererence.

In this work, we will have the same goal, as on to improve the original BiDAF model with respect to accuracy, train/infer time and extensibility.

3 Model Overview

The high level Architecture of the model (as described in Fig. 1) is very similar to the tradition approaches, it is a hierarchical multi-stage process consisting of 6 layers: Character Embedding Layer, Word Embedding Layer, Contextual Embedding layer, Attention Flow layer, Modeling layer & Output layer.

In Detail, the layers performs the below tasks:

1. **Character Embedding Layer:** Both the Original BiDAF implementation and the subsequent variants of it uses the same approach. The character embedding is obtained by using Convolution Neural Network as proposed by Kim (2014) [11]. Specifically, Characters are embedded into vectors, which can be considered as 1D inputs to the CNN, and whose size is the input channel size of the CNN. The outputs of the CNN are max-pooled over the entire

width to obtain a fixed-size vector for each word. Therefore, Each character is represented as a trainable vector.

Our Approach is different from the existing approaches, We use the existing pretrained word embeddings (glove 840B 300d [12]) to extrapolate the corresponding character embedding. We iterate through all the characters present in the Glove word embedding corpus and extract it's parent words, ie. glove words that consist the respective character. Then, we perform a simple averaging of these parent word embeddings. The resulting character embedding has been found to be effective [13] [14].

2. **Word Embedding Layer:** Our approach is same as the previous ones, We use pre-trained word vectors, GloVe (Pennington et al., 2014), to obtain the fixed word embedding of each word.

   The concatenation of the character embedding and the word embedding is passed to a two-layer Highway Network (Srivastava et al., 2015). The output is a d-dimensional vector representing the sequence input.

   We slightly divert a little during computation of character embedding for a sequence. To get a vector representation for a token in a sequence, we average their character embedding instead of using CNN.

3. **Contextual Embedding Layer:** This layer is responsible for modeling the temporal interactions between words. The original BiDAF implementation used Bi-directional LSTM [15] to perform this. We replace this with transformer encoder [6]. Along with the traditional transformer encoder, we add few enhancements to optimize them for better performance. Refer Fig. 2.

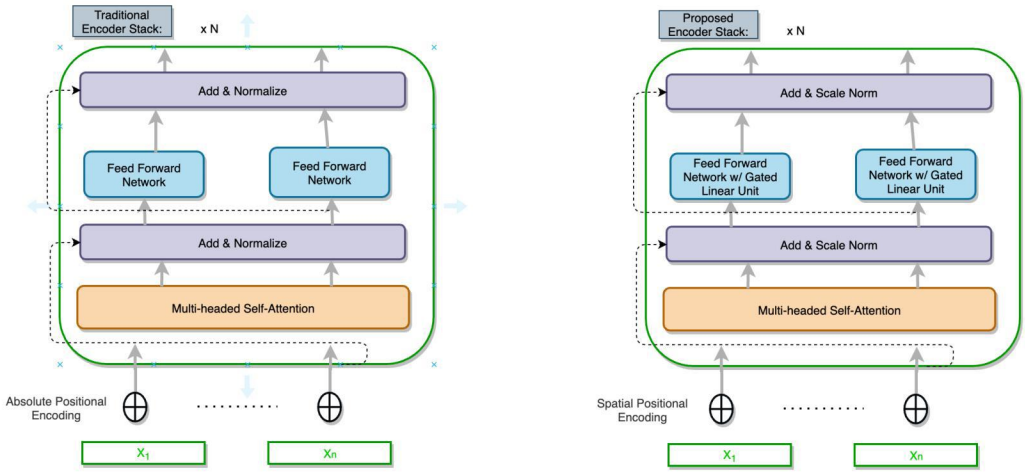

Figure 2: An Overview of the traditional encoder stack (left) and the Overview of our encoder stack (right).

Let's look at the modifications one-by-one:

(a) **Spatial Positional Encoding:** A positional encoding is a finite-dimensional representation of the location or "position" of items in a sequence. Given some sequence A = [a_0, . . . , a_n-1], the positional encoding must be some type of tensor that we can feed to a model to tell it where some value a_i is in the sequence A. The traditional way of doing this is by infusing fixed positional encoding based on the index of the token in the sequence. We implement a recent alternate approach which showed very good results.

Independently introduced by two authors [10] [9], This technique injects fixed sinusoidal position embedding into the input prior to their projection. The Positional embedding is added to the query and the Key vector at each layer. The outputs at each layer are the transformed, weighted sums of the value vectors.

(b) **Scale Norm:** Introduced by Toan Q. Nguyen et al. [8], This technique proposes to replace the traditional Layer Norm with scaled l2 Normalization. Authors explain that

this is similar to projecting d-dimensional vectors on-to a (d-1) dimensional hypersphere with learned radius g. The results from the paper show that this technique leads to faster convergence.

$$ScaleNorm(x; g) = g\frac{x}{||x||} \tag{1}$$

(c) **FeedForward Network with Gated Linear Unit:** Introduced by Noam Shazer [7], The technique introduces Gated Linear Units that consist of the component-wise product of two linear projections, one of which is first passed through a RELU function. Specifically, The Traditional FFN takes a vector x (the hidden representation at a particular position in the sequence) and passes it through two learned linear transformations and a RELU activation function is applied between these two. Refer Fig. 3 for comparison of different approaches.

$$\begin{aligned}
\mathrm{FFN}_{\mathrm{GLU}}(x, W, V, W_2) &= (\sigma(xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{Bilinear}}(x, W, V, W_2) &= (xW \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{ReGLU}}(x, W, V, W_2) &= (\max(0, xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{GEGLU}}(x, W, V, W_2) &= (\mathrm{GELU}(xW) \otimes xV)W_2 \\
\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) &= (\mathrm{Swish}_1(xW) \otimes xV)W_2
\end{aligned}$$

Figure 3: comparable equations from the original paper.

4. **Attention flow Layer:** Both the Original BiDAF implementation and the subsequent variants of it builds a similarity matrix between each pair of context and query words. The model uses C and Q to denote the encoded context and query. The context-to-query attention is constructed as follows: We first compute the similarities between each pair of context and query words, rendering a similarity matrix $S \in R^{n*m}$. We then normalize each row of S by applying the softmax function, getting a matrix $S^{|}$. Then the context-to-query attention is computed as $A = S^{|} \cdot Q^t \in R^{n \cdot d}$. The similarity function used here is the trilinear function (Seo et al., 2016):

$$f(q, c) = W_0[q, c, q \circ c] \tag{2}$$

Our Approach is very simple, and similar to how transformer decoder network computes cross Attention between decoder hidden state and encoder hidden state, we computes cross attention between encoded context embedding and encoded question embedding. More specifically, we compute query vector (here, we refer encoder query representation as query vector and the encoded representation of SQuAD query as question vector) from encoded context embedding. We compute key and value vector from encoded question embedding. After this, we perform our usual multi-headed self-attention mechanism. The size of the final output would be (batch size, context seq len, hidden size)

we then concatenate the encoded context representation and the cross-attention output.

$$f(q, c) = [c, crossAttention(q, c)] \tag{3}$$

5. **Modelling Layer:** Our Modeling Layer contains 3 stacked encoder blocks(Refer Fig. 2) with 8 heads.

6. **Output Layer:** Our Output layer is more similar to QANet implementation than the original BiDAF model. We use the encoded representation of the cross attention from modelling layer to compute start token and end token probabilities. For start token, We do a linear transformation of the encoded vector and then take a log softmax. For the end token, We pass the encoded vector to another shallow encoder network and the resultant vector is passed through a log softmax.

4 Experiments

4.1 Data

Our Dataset is the modified version of official SQuAD 2.0 Dataset, that was provided to us for this default project. The Dataset is split into train, dev and test set. Where the train set is exactly the same as the official train-set but the dev and test set is a sample of the office dev set. The train and dev set consist of (context, question, answer) triples.

4.2 Evaluation method

We use the same metrics as the official SQuAD leaderboard, that is, exact match (EM) and F1 Score. Exact Match measures whether our answer span exactly fits with the annotated answer span, whereas F1 score takes partial scores by computing the harmonic mean of precision and recall, where precision is calculated as the number of correct words divided by the length of the predicted answer, and recall is calculated as the number of correct words divided by the length of the ground truth answer.

$F1 = 2 \cdot \left(\frac{precision \cdot recall}{precision + recall} \right)$

4.3 Experimental details

For our Baseline model, we extend it to include char embedding as proposed in the original BiDAF model (from here on, when we refer to baseline model, it refers to this model with char embedding included). We train the model with a learning rate of 5e-1, batch size of 128, hidden size of 100 (100 word dim & 100 char dim concatenated in Contextual embedding layer), dropout of 0.1 and We use AdamW as the optimizer, and a scheduler to include a warm-up period of 3 epochs and reduce the learning rate linearly till 25 epoch after the warm-up period.

For our subsequent attention models, we change the learning rate from 5e-1 to 1e-3, batch size of 32, hidden size of 100, dropout of 0.2 and a warm-up period of 10 epochs. For our encoders and cross-attention, the number of stacked layers (depth) and the number of heads in Embedding layer, attention layer and Modelling layer is 1:3, 3:12, 1:8, respectively.

The details are fully summarized in Fig. 2

Models	Baseline	Baseline+char	baseline+char+crossAttention	baseline+char+crossAttention+encoders
Batch size	128	128	128	32
Hidden size	100	100	100	100
learning rate	5e-1	5e-1	1e-3	1e-3
warmup-period.	3 epochs	3 epochs	10 epochs	10 epochs
L2 decay rate	0.001	0.001	0.001	0.001
eps	1e-8	1e-8	1e-8	1e-8
Embedding Layer depth	1	1	1	1
Embedding Layer Attentions heads	-	-	-	3
Attention Flow Layer depth	1	1	3	3
Attention Flow Layer Attention heads	-	-	12	12
Modelling Layer depth	2	2	2	1
Modelling Layer Attentions heads	-	-	-	8
Epochs trained on for comparison	8	8	8	8

Table 1: Experimental details for various models

We initially wanted our model to run with the batch size 128, hidden size 300 and encoders with more depth but that was too much for our compute resource, therefore we decided to experiment with the simpler version of it.

Other parameters of the model are very standard, Adam optimizer with beta value of (0.8, 0.999), an moving average on all training variables with a decay rate of 0.999. We used 0.1 dropout in our embedding layers and 0.2 elsewhere.

The maximum sequence length for our encoders were kept at 512 since 400 was the maximum answer token length we saw in the training set and the query lengths were less than 25 with few exceptions.

We trained all of the models to 8 epochs and compared their performance. This was mainly due to time and resource constrain. We believe, with more resource and training time, capacity of our model will be much greater than the original model due to their bottleneck of RNN. In most of the cases,

we were able to infer patterns from the metric graph (F1 score, NLL loss on dev & train set) that easily explains whether the model is still learning (positive line) or saturated (flat line) or degrading (negative line).

4.4 Results

Models	F1	EM	AvNA
Baseline BiDAF (with char)	55.44	52.01	62.66
BiDAF + Pretrained Char	58.77	55.59	65.55
BiDAF + Pretrained Char + CrossAttention	59.66	56.67	67.21
BiDAF + Pretrained Char + CrossAttention + encoders	**64**	**60.85**	**70.01**

Table 2: Comparison of Models' performance on Dev set trained for approximately 1M steps

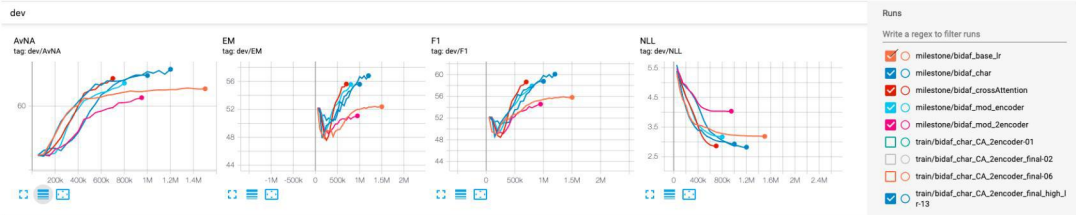

Figure 4: Our model scores charted at each iteration.

As we can see from the table 2, the Pre-trained Character embedding really helps the model, and every enhancement we added, the model score improved. We didn't expect the final model scores to be that close to the Baseline BiDAF approach but considering we built a very shallow encoder and used only 100 dim, it doesn't suprise us that much. With enough compute resource, our model should easily outperform the BiDAF and maybe the QANet model.

From the Fig 4, we can observe that the base BiDAF model is being outperformed by all other approaches and our BiDAF_crossAttention (BiDAF + char + attention) model hasn't even started to degrade in it's learning yet. Our fourth model BiDAF_mod_encoder was an experimental model which used the original BiDAF attention flow layer and along with it, included encoders instead of RNN's. This model didn't perform as expected, majorly due to the shallowness of our encoder. with few more extra depth and heads, the model should be able to perform way better than RNNs.

During our Experiments we found lot of observations that are worthy to mention.

- **Adding a warm-up period to the optimizer** made a huge difference in training time and it led the model to faster convergence. In our case, as the model has not been pre-trained, it helps to reduce the impact of deviating the model from learning on sudden new data set.
- **Learning rate makes all the difference**. As we were on the clock, we initially wanted to fix the learning rate and experiment with other parameters like hidden size, encoder depth, number of heads in attention, etc. But it took a lot of time for us to just figure out the appropriate learning rate for RNNs and transformer encoders. We didn't have enough time to train the model with multiple learning rates, therefore we implemented something similar to Cyclic LR [16] (Leslie N. Smith, 2015) with decreasing amplitude and set a high learning rate like 0.5 and observed the NLL and it's variance over the course of iterations.
- Except the difference in training and inference time, **we didn't observe that much difference with swapping out RNN's for encoders**. At least, replacing them with shallow encoders didn't help. You can observe this in Fig. '4, where the bidaf+mod+2encoder model didn't exactly top off the chart. But we definitely think that increasing the encoder parameters like depth and number of heads would improve the scores.
- **Multi-head Attention is all you need!**. When we initially designed the approach for Attention Flow Layer, we weren't fully convinced that it would outperform the original implementation but the more experiments we did, we came to know that multi-headed

attention really helps to capture the context-to-query representation, and adding more heads is beneficial than increasing the depth.

5 Analysis

We didn't have enough time to do a complete Qualitative Analysis on our models as our experiments took a big chuck of our allotted time (Our non-encouraging Experiments are listed in the Appendix). But we iteratively looked at the results our model was producing and tried to adapt our parameters by what we observed.

Below are our observations:

- The model performed extremely well on the answers to the "when" questions. On the other hand, answers to the "Why" questions were poor. The model was very likely to produce "No Answer" to these questions.
- We found that majority of the answers in our dataset are of word length 5 or less. Answer word length of more than 8 are very low. When we looked at the answers for these questions (for which true answers of length more than 5), the model either provided shorter answers (span of the true answer) or gave "No Answer". This shows that our model modelled the length of answers and as in our case, prefers answers with shorter length due to our dataset. One experiment which we wanted to do but didn't do due to time constrain was to penalize the model when the difference of the predicted answers length and the true answers length is more. Another solution would be to augment the data by back-translation or text-generation.
- Based on the few examples we observed, we were able to say that the model lacked semantic context at times. We quickly hypothesised that this was due to our pre-trained feature embedding transformation. As our hidden size is only from 64-100. We transform our embedding of size 300 dimension to, say 100 dimension. This reduces the quality of the embeddings. The solution for this would be to either use higher hidden dimension or fine-tune an word embedding on our paragraph (context) data.
- We observed our results to see how adding pre-trained character embedding helps and we found that it was really helpful when the words are used of regular context or it's OOV.

6 Conclusion

In this project, we have explored the underlying problems of BiDAF, and explored the approach to enhance it with latest research techniques like encoders, multi-headed attention, etc. We started with the Baseline BiDAF model and then enhanced it by utilizing some of the mentioned techniques. We have built encoders, cross-attention, spatial positional encoding, etc. from scratch.

We show that our modified simple end-to-end network performs better than the original BiDAF approach and has lots of room for improvement in the performance when model capacity is increased.

We further wish to improve this model in the future by improving the modelling layer and the output layer. Additionally, approaches like Conditioning our end token based on the start token would help the model to predict better.

7 Acknowledgements

Our Transformer implementation was hugely inspired by the work of Phil Wang [17]. We followed his coding style, function signatures and used his implementation as reference to implement our own from scratch. Massive thanks to him for this. As Always, we are very thankful for CS224n course instructors and teaching assistants. An extra shout-out to azure for providing us with compute resources. We had some issues on running our final model in the test set leaderboard, we hope to quickly resolve this and share the results.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[5] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, Mar 2020.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[7] Noam Shazeer. Glu variants improve transformer, 2020.

[8] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth, 2020.

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[10] Ofir Press, Noah A. Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs, 2020.

[11] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[12] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[13] Pretraineddissecting google's billion word language model part 1: Character embeddings.

[14] Pretrained character embeddings for deep learning and automatic text generation.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[16] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[17] Phil Wang. Transformers. `https://github.com/lucidrains`.

[18] Mittens. Mittens. `https://github.com/roamanalytics/mittens`.

[19] Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory, 2019.

A Appendix

We wanted to list down few other experiments we performed which didn't work out very well.

- **Language model training** We build a simpler version of the popular BERT LM [3]. We used the 2 training tasks masked language modelling (MLM) and Textual Entailment to train the model on our SQuAD dataset (only on context paragraph). Specifically, we extracted the context paragraph from all the samples in our train data-set and used that for feature training. We later added a simple classification head to predict the start-token and end-token.
  The model had high MLM loss and perplexity. Therefore, we hypothesised that the model is unable to learn the context due to the small training data, which seemed very likely and reasonable explanation. Still, our final F1 score on SQuAD dev dataset was 47.

- **Fine-tuning glove word vectors** We tried fine-tuning our word-vectors using an open-source implementation (mittens [18]). The results weren't very encouraging, therefore we dropped it.
- **Augmenting Self-attention with Persistent Memory** We tried to implemented a new approach introduced by Sainbayar et el [19]. The technique was to add learned memory key/value prior to the attention mechanism. We assumed we were a little out-of-depth on this theoretical implementation as the results which we received were not encouraging.